



# FITS Data Format for Calibrated Imaging Data

Tom Pauls, NRL

John Young, Univ. Cambridge

<http://www.mrao.cam.ac.uk/~jsy1001/exchange/>



# Acknowledgements

- Bill Cotton
- David Buscher
- John Monnier
- Dave Mozurkewich
- Tom Armstrong
- Pascal Ballester
- Nick Elias
- Thierry Forveille
- Gilles Duvert
- Andy Boden
- Those at AAS198 round table discussion



# Outline

- History
- Why?
- Current status
- Format description
  - Scope
  - FITS Primer
  - File structure
  - Software
- Current and planned implementations



# History

- Idea from June 2000 NSF-sponsored meeting in Socorro
- Since 2001, TP & JSY + community have been working on the format
- Drafts discussed at
  - AAS 198, June 2000
  - IAU WG Meeting, August 2001
- Public pre-release, March 2002
  - document + example code
- IAU WG Meeting, Kona, August 2002
- Version 1.0 frozen, April 2003



# Why a common data format?

- Share data
  - With collaborators (combine datasets)
  - End-users of facility arrays
- Share software
  - Save effort
  - Get better software



## Current status

- Version 1.0 release!
- Specification document
- Example C code (cfitsio)
- John Monnier's IDL reader/writer code
- John Young's Python code

See

<http://www.mrao.cam.ac.uk/~jsy1001/exchange/>



## What it is/isn't

- Aimed at imaging applications
- Calibrated (and averaged) data only
- Not astrometry (yet?)
- Not space interferometry
- Not an archive format
  - Minimalist

But FITS is flexible – you can add extra tables



# FITS primer

- Originally for images only. Now much more flexible
- Sequence of Header + Data units (HDUs):
  - ASCII header: **keywords** & values
  - Data, one of:
    - Image
    - ASCII Table (not in 1<sup>st</sup> HDU)
    - Binary Table (not in 1<sup>st</sup> HDU)
- We only use **binary tables**:
  - Columns (with **headings**)
  - Values in columns may be scalars or vectors
  - Many data types (integer, real, complex)
  - Table structure described by mandatory keywords in header
- Relational database



# Example bintable header

```
XTENSION= 'BINTABLE'                      / binary table extension
BITPIX   = 8                                / 8-bit bytes
NAXIS    = 2                                / 2-dimensional binary table
NAXIS1   = 46                               / width of table in bytes
NAXIS2   = 4                                / number of rows in table
PCOUNT   = 0                                / size of special data area
GCOUNT   = 1                                / one data group (required keyword)
TFIELDS  = 5                                / number of fields in each row
TTYPE1   = 'TEL_NAME'                       / label for field 1
TFORM1   = '8A'                             / data format of field: ASCII Character
TTYPE2   = 'STA_NAME'                       / label for field 2
TFORM2   = '8A'                             / data format of field: ASCII Character
TTYPE3   = 'INDEX'                          / label for field 3
TFORM3   = 'I'                              / data format of field: 2-byte INTEGER
TTYPE4   = 'DIAMETER'                       / label for field 4
TFORM4   = 'E'                              / data format of field: 4-byte REAL
TUNIT4   = 'm'                             / physical unit of field
TTYPE5   = 'STAXYZ'                         / label for field 5
TFORM5   = '3D'                            / data format of field: 8-byte DOUBLE
TUNIT5   = 'm'                             / physical unit of field
EXTNAME  = 'OI_ARRAY'                      / name of this binary table extension
REVISION= 0                                / Revision number of the table definition
ARRNAM   = 'COAST'                          / Array name
FRAME    = 'GEOCENTRIC'                     / Coordinate frame
ARRAYX   = 3920635. / [m] Array centre x coordinate
ARRAYY   = -2889. / [m] Array centre y coordinate
ARRAYZ   = 5013987. / [m] Array centre z coordinate
DATE-OBS= '2000-10-19'                     / Start date of observations
END
```



# Example bintable data

Displayed by **fv** (part of ftools):

X-fv : Binary Table of testdata.fits[1] in /home/sy1001/src/exchange

File Edit Plot Sort Calc Histogram Statistics Help

TEL\_NAMI  STA\_NAMI  INDEX  DIAMETER  STAXYZ  
8A 8A I E 3D  
m m

expand

1	TEL2	C	1	4.0000E-01	0.0000000000E+00
2	TEL4	W4	2	4.0000E-01	4.4740000000E+00
3	TEL3	E3	3	4.0000E-01	3.6320000000E+00
4	TEL1	N3	4	4.0000E-01	-1.1240000000E+01

Go to:  Edit cell:



## Defined tables

- Separate binary table for each data type:
  - Squared visibility (OI\_VIS2)
  - Complex visibility (OI\_VIS)
  - Triple product (OI\_T3)
- Accompanying OI\_WAVELENGTH table
- Data tables include definitive  $u$ ,  $v$  coordinates
- Table with minimal target information (OI\_TARGET)
- Table giving info needed to reproduce/interpolate  $u$ ,  $v$  coordinates (OI\_ARRAY)



## Undefined tables/features

- Equivalents to OI\_ARRAY for aperture masking, orbiting arrays etc.
- Instrument-specific information
- Astrometric information (internal metrology etc.)
- Polarisation state
- Description of  $u, v$  plane averaging



## Using the example code

Example routines read/write from/to data structures (C structs) in memory. These structures mimic the file structure as far as possible.

Two ways to use the routines:

1. Use routines as is, your code interfaces to the data structures provided.
2. Create equivalent I/O routines that read/write from/to more appropriate data structures.



```
/**  
 * Write OI_WAVELENGTH fits binary table  
 *  
 * @param fptr      see cfitsio documentation  
 * @param wave      wavelength data struct, see exchange.h  
 * @param status    pointer to status variable  
 *  
 * @return On error, returns non-zero cfitsio error code, and sets status  
 */  
int write_oi_wavelength(fitsfile *fptr, oi_wavelength wave, int *status) {  
  
    const char function[] = "write_oi_wavelength";  
    const int tfields = 2;  
    char *ttype[] = {"EFF_WAVE", "EFF_BAND"};  
    char *tform[] = {"E", "E"};  
    char *tunit[] = {"m", "m"};  
    char extname[] = "OI_WAVELENGTH";  
    int revision = 0;  
  
    if (*status) return *status; /* error flag set - do nothing */  
    fits_create_tbl(fptr, BINARY_TBL, 0, tfields, ttype, tform, tunit,  
                   extname, status);  
    if (wave.revision != revision) {  
        printf("WARNING! wave.revision != %d on entry to write_oi_wavelength.  
Writing revision %d table\n", revision, revision);  
    }  
    fits_write_key(fptr, TINT, "REVISION", &revision,  
                  "Revision number of the table definition", status);  
    fits_write_col(fptr, TFLOAT, 1, 1, 1, 1, wave.eff_wave, status);  
    fits_write_col(fptr, TFLOAT, 2, 1, 1, 1, wave.eff_band, status);  
  
    if (*status) {  
        fprintf(stderr, "CFITSIO error in %s:\n", function);  
        fits_report_error(stderr, *status);  
    }  
    return *status;  
}
```



## Other software

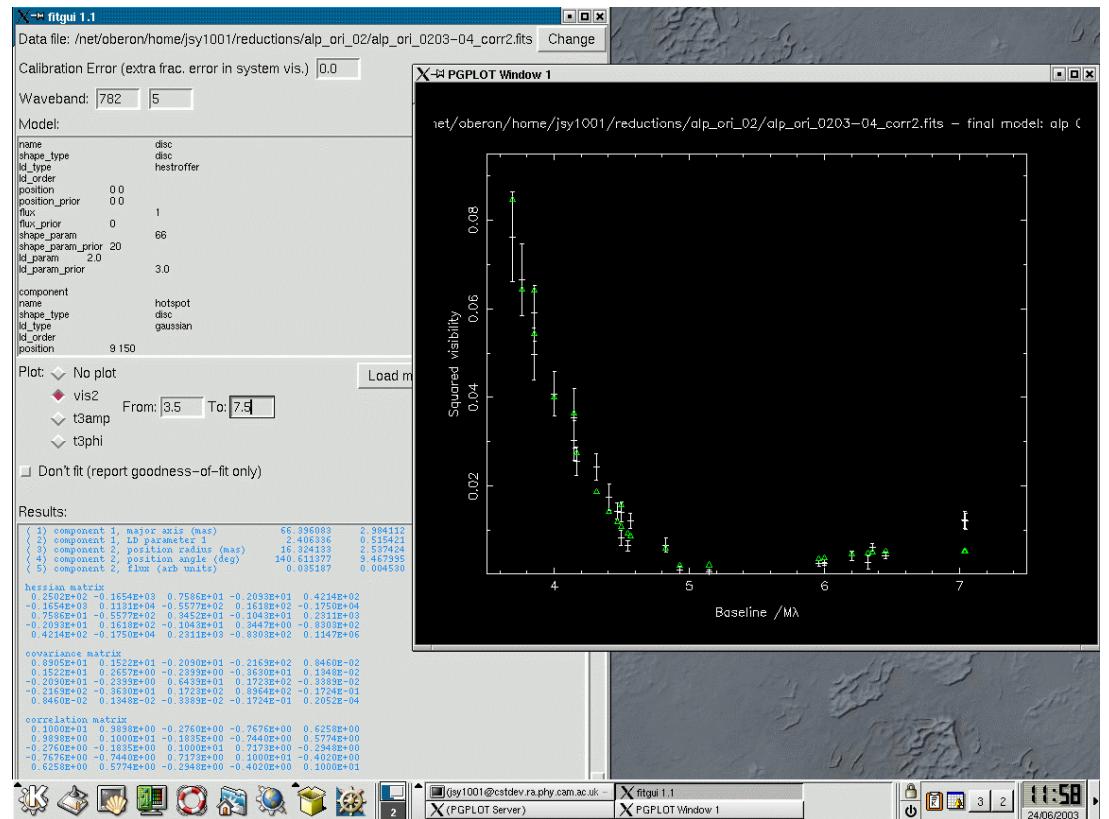
- Completed
  - IDL reader/writer (John Monnier)
    - OYSTER (USNO)
- Under construction
  - AIPS++ interface (NRL)
  - Model-fitting software (Univ. Cambridge)
  - BSMEM (Univ. Cambridge)



# mfit (Univ. Cambridge)

## Bayesian model-fitting to OI-FITS-format data

- Asymmetric, multi-component models
- Four limb-darkening parameterizations
- Interactive, command-line, and GUI interfaces
- Plots of data & model





## Current and Planned Implementations

- COAST
- NPOI
- ISC
- VLTI
- Others?



# Conclusion

**Use it!**